Distributed Computing syllabus, focusing on **Distributed Database Systems (DDBS)**.

Unit-1.0: Concept and Overview of Distributed Database System 🌐

What is a Distributed Database System (DDBS)?

A **Distributed Database System (DDBS)** is a collection of multiple, logically interrelated databases distributed over a computer network. The system appears to the user as a single, centralized database. The data is stored across several physical sites, and the management of this data is handled by a **Distributed Database Management System (DDBMS)**.

Features of DDBS

- **Distributed Data Storage:** Data is stored across multiple sites.
- **Logical Interrelation:** The data at the different sites are logically connected and represent a single logical database.
- **Network Connection:** The sites are connected via a computer network.
- **Distributed Transaction Management:** Transactions can access and update data across multiple sites.
- **Location Transparency:** Users do not need to know where the data is physically stored.

Promises (Advantages) of DDBS

- **Increased Reliability and Availability:** If one site fails, the data may still be accessible from other sites (due to replication).
- **Improved Performance:** Queries can be executed in parallel at multiple sites, and data is stored closer to where it's most frequently used.
- **Easier System Expansion:** New sites and data can be added without affecting the operations of existing ones (modularity).
- **Better Economics:** The cost of interconnecting smaller computers can be more cost-effective than using a single, large mainframe.
- **Data Sharing:** Data can be shared among different users and applications across the network.

Design Issues in DDBS

- **Transparency:** Providing a view of a single database despite physical distribution (e.g., location, fragmentation, and replication transparency).
- **Distributed Query Processing:** Optimizing queries that access data across multiple sites.
- **Distributed Concurrency Control:** Ensuring consistency when multiple transactions access distributed data simultaneously.
- **Distributed Deadlock Management:** Detecting and resolving deadlocks that span multiple sites.
- **Distributed Database Recovery:** Restoring the database to a consistent state after failures (e.g., site crashes, network partitions).
- **Data Fragmentation and Allocation:** Deciding how to split the data and where to store the fragments.

Distributed DBMS Architecture

1. Client/Server System

- **Structure:** This is a two-tier architecture. **Clients** are user applications that submit queries/transactions. **Servers** manage the database and execute transactions.
- **Interactions:** Clients send requests to servers, and servers return the results. A client might interact with multiple servers, and a server might service multiple clients.
- **Benefit:** Good separation of concerns and scalability.

2. Peer-to-Peer (P2P)

- **Structure:** All sites (nodes) are considered equals, acting as both clients and servers. There is no central control.
- **Interactions:** Any node can initiate a transaction and access data at any other node. Nodes cooperate to process transactions.
- **Benefit:** High autonomy and fault tolerance.

3. Multi-Database System (MDBS)

- **Structure:** A collection of pre-existing, independent, and possibly **heterogeneous** local databases. A global layer (the MDBS) provides an integrated view.
- **Interactions:** Local DBMSs retain their autonomy. The MDBS layer translates global queries into sub-queries for the local DBMSs.

- **Benefit:** Allows the integration of existing, disparate databases without requiring them to be redesigned.

Unit-2.0: Distributed Database Design 📐

Distributed Database Design Concept

This process involves two main phases: **data distribution** (how to partition and replicate data) and **schema design** (defining the structure of the fragments). The goal is to maximize performance, availability, and reliability while minimizing communication costs.

Objective of Data Distribution

The primary objectives are to:

1. **Improve Performance:** Place data near its primary user (minimizing remote access).
2. **Increase Availability and Reliability:** Replicate critical data so it's accessible even if a site fails.
3. **Ensure Scalability:** Allow the system to grow by adding new sites and data.

Data Fragmentation

The process of dividing a global relation (table) into smaller, logical units called **fragments**.

- **Horizontal Fragmentation:** Dividing a relation into subsets of **rows** (tuples). Each fragment has the same columns but different rows.
    - *Example:* Splitting an 'Employee' table by city.
- **Vertical Fragmentation:** Dividing a relation into subsets of **columns**. Each fragment has different columns but the same primary key to allow for reconstruction.
    - *Example:* Splitting an 'Employee' table into one fragment with personal details and another with salary details.
- **Mixed Fragmentation:** A combination of horizontal and vertical fragmentation.

The Allocation of Fragments

This is the process of deciding **where** to physically store the fragments across the network sites.

- **Non-replicated (or Centralized) Allocation:** Each fragment is stored at exactly one site. (Used in centralized systems and some simple DDBSs).

- **Replicated Allocation:**
    - **Fully Replicated:** A copy of every fragment is stored at **every** site. (High availability, but high update cost).
    - **Partially Replicated:** A fragment is stored at **some**, but not all, sites. (The most common approach, balancing availability and update cost).

Transparencies in Distributed Database Design

Transparency hides the details of the distributed nature of the system from the user, making it seem like a single, centralized database.

Type of Transparency	What it Hides
Data (Location) Transparency	The site where the data is physically stored.
Fragmentation Transparency	The fact that a relation is divided into fragments.
Replication Transparency	The fact that copies of data/fragments exist at multiple sites.
Network Transparency	The details of the communication network.
Transaction Transparency	The fact that a transaction may execute across multiple sites.

Export to Sheets

Unit-3.0: Distributed Transaction and Concurrency Control 🔒

Basic Concept of Transaction Management

A **transaction** is a sequence of database operations (like Read, Write, Insert, Delete) that is treated as a single, indivisible unit of work. They must adhere to the **ACID properties**:

- **Atomicity:** All operations in the transaction must be completed, or none must be completed (all or nothing).
- **Consistency:** The transaction must bring the database from one valid state to another.
- **Isolation:** The effect of a transaction must be invisible to other concurrent transactions until it is committed.
- **Durability:** Once a transaction is committed, its changes are permanent, even in the event of a system failure.

Objective of Distributed Transaction Management

The core objective is to ensure the **ACID properties** are maintained even though a single transaction might execute over multiple, independent sites.

Model for Transaction Management

A global transaction is typically decomposed into a set of independent **sub-transactions**, one for each site that holds data the transaction needs to access.

- **Transaction Coordinator (Global):** Responsible for initiating the sub-transactions, monitoring their execution, and coordinating the commit or abort process across all sites.
- **Local Transaction Managers:** At each site, they manage the sub-transactions, local recovery, and local concurrency control.

Distributed Concurrency Control (DCC)

The mechanisms used to synchronize concurrent access to distributed data to maintain database consistency (Isolation property).

Objective of DCC

To achieve **global serializability**, meaning the concurrent execution of distributed transactions must produce the same result as some serial execution of those same transactions.

Concurrency Control Anomalies (Problems)

If DCC is not enforced, anomalies can occur:

1. **Lost Update:** One transaction's update is overwritten by another's.
2. **Dirty Read:** A transaction reads data written by another uncommitted transaction.
3. **Non-repeatable Read:** A transaction reads the same data item twice and gets different values because another committed transaction updated it in between.

Distributed Serializability

A schedule of distributed transactions is **serializable** if it is equivalent to some serial schedule of those same transactions. This is the correctness criterion for concurrency control in DDBS.

Locking Based Algorithm (Two-Phase Locking - 2PL)

The most common approach.

- **Basic Idea:** A transaction must acquire a lock on a data item before accessing it (shared lock for read, exclusive lock for write).
- **Two Phases:**
    1. **Growing Phase:** A transaction can only acquire locks, not release any.
    2. **Shrinking Phase:** A transaction can only release locks, not acquire any new ones.
- **Distributed 2PL:** The global coordinator coordinates lock requests. If a transaction needs a lock on a replicated item, it must acquire the lock on all copies (or a majority, depending on the protocol).

Unit-4.0: Distributed Deadlock and Recovery ♻️

Introduction to Deadlock

A **deadlock** is a state where two or more transactions are waiting indefinitely for resources (locks) held by each other.

- *Example:* Transaction holds lock and waits for . Transaction holds lock and waits for .

Distributed Deadlock Prevention

Ensuring that at least one of the four necessary conditions for deadlock (mutual exclusion, hold and wait, no preemption, circular wait) can never occur.

- **Wait-Die:** An older transaction () is allowed to wait for a resource held by a younger transaction (). If requests a resource held by , waits. If requests a resource held by , is aborted (dies) and restarts later.
- **Wound-Wait:** A younger transaction () is allowed to wait for an older transaction (). If requests a resource held by , waits. If requests a resource held by , is preempted (wounded/aborted) and restarts later.

Distributed Deadlock Avoidance

Requires transactions to declare their required resources in advance, which is impractical in most DDBS.

Distributed Deadlock Detection and Recovery

1. **Wait-For Graph (WFG):** A directed graph where nodes are transactions and an edge means is waiting for a resource held by . A **deadlock exists if and only if there is a cycle** in the WFG.

2. **Detection Strategies:**
    - **Centralized:** A single site maintains the global WFG. Simple, but a single point of failure.
    - **Hierarchical:** WFGs are managed in a tree structure.
    - **Distributed (e.g., Phantom Detection/Chandy-Misra-Haas):** All sites cooperate. Sites send special **probe messages** to detect cycles in the WFG fragments distributed across sites.
3. **Recovery:** Once a deadlock is detected, a transaction (the **victim**) is chosen and aborted to break the cycle.

Two-Phase Commit (2PC) Protocol

A mandatory atomic commitment protocol used for distributed transactions to ensure **Atomicity** across all participating sites.

1. **Phase 1: The Voting Phase (Preparation)**
    - The **Coordinator** sends a **VOTE_REQUEST** (prepare) message to all participating **Participants**.
    - Each Participant attempts to complete its part of the transaction and responds with either:
        - **VOTE_COMMIT** (Ready to Commit) - It has successfully prepared and saved all necessary data/logs.
        - **VOTE_ABORT** (Not Ready) - It failed or chose to abort.
2. **Phase 2: The Decision Phase (Commit/Abort)**
    - **If all Participants sent VOTE_COMMIT:** The Coordinator sends a **GLOBAL_COMMIT** message. All Participants make their changes permanent.
    - **If at least one Participant sent VOTE_ABORT or failed to respond:** The Coordinator sends a **GLOBAL_ABORT** message. All Participants undo their changes.

Three-Phase Commit (3PC) Protocol

An extension of 2PC designed to eliminate the blocking problem (where a coordinator failure after Phase 1 of 2PC can leave participants blocked).

- **Phases: Prepare**, **Pre-Commit**, and **Commit**.

- **Mechanism:** It adds a **Pre-Commit** phase to ensure all sites know the unanimous decision before the final commit is sent, reducing the chance of blocking if a failure occurs. However, it is more complex and introduces its own set of issues (e.g., blocking during network partition).

Unit-5.0: Distributed Query Processing and Optimization 🚀

Concepts, Objective, and Phases of Distributed Query Processing

Concept

Processing a query that requires accessing and combining data located at multiple, distinct sites. This involves moving data between sites, which is the dominant cost factor.

Objective

To find an execution strategy that **minimizes the total cost** (primarily communication/data transfer costs, but also I/O and CPU costs).

Phases of Distributed Query Processing

1. **Query Decomposition (Local):** The global query is parsed, validated, and decomposed into an initial relational algebra query.
2. **Data Localization (Global):** The global query is rewritten into fragments using information about data fragmentation and replication. This often involves replacing global relations with joins/unions of their fragments.
3. **Global Query Optimization:** An optimal execution plan is selected based on a cost model, focusing on minimizing inter-site data transfer. This plan specifies the site where operations (like joins) will be performed.
4. **Local Query Optimization:** At each site, the sub-query is optimized for local execution (minimizing local I/O and CPU costs).

Join Strategies in Fragment Relation

Joining relations whose fragments reside at different sites is the most critical and expensive operation. The goal is to reduce the amount of data transferred.

- **Semijoin:** A technique to reduce the size of the relation that needs to be transferred.
    1. Project the joining attribute(s) of one relation () at its site ().
    2. Send this projection to the site of the other relation ().

    3. Select tuples in the second relation () whose joining attributes match the received projection.
    4. Send the reduced relation back to .
    - *Operation:* . It's highly effective when is much larger than .

- **Hybrid Join:** Combines local processing with data transfer. For example, performing a local join of fragments at one site and then shipping the result to another site for the final join.
- **Broadcast Join:** If one relation () is small, it can be broadcast to all sites that hold fragments of the other relation (). The joins are performed locally at those sites.

Global Query Optimization

The process of determining the best execution strategy for the distributed query, primarily by choosing the optimal sequence and location for distributed joins. The optimizer typically considers factors like:

- **Relation/Fragment Sizes:** The number of tuples in each fragment.
- **Selectivity of Predicates:** How much a filter (WHERE clause) reduces the size of a fragment.
- **Transmission Cost:** The cost of moving data between sites.
- **Available Network Bandwidth:** The speed of data transfer.

Unit-6.0: Heterogeneous Database 🧩

Architecture of Heterogeneous Database (HDB)

A **Heterogeneous Database System (HDBS)** is an MDBS where the local DBMSs use different data models (e.g., Relational, Object-Oriented, Network) or different software (e.g., Oracle, SQL Server, MySQL).

- **Components:**
    1. **Local Database Systems (LDBS):** Autonomous, existing database systems.
    2. **Local Schemas:** The native schema of each LDBS.

3. **Local Wrappers/Gateways:** Software components that translate the local data model and query language into a common model (canonical model) for the integration layer.
4. **Global Schema (Integrated Schema):** A single, unified, conceptual schema that integrates the data from all LDBSs.
5. **Global Query Processor:** The component that accepts global queries, translates them into the canonical model, and decomposes them into sub-queries for the LDBSs.

Database Integration

The core challenge in HDBS is integrating the disparate local schemas.

1. Schema Translation

- **Goal:** Convert the schema of a local database from its native data model (e.g., XML, Network) into a common, canonical data model (often Relational or Object-Oriented).
- **Process:** Handled by the **wrapper** at each site. This is necessary so that the global system can view all local data in a unified format.

2. Schema Integration

- **Goal:** Combine the individual, translated local schemas into a single, consistent, and non-redundant **Global Schema**.
- **Issues (Integration Conflicts):**
    - **Naming Conflicts (Synonyms/Homonyms):** Different names for the same entity (**Synonyms**) or the same name for different entities (**Homonyms**).
    - **Structural/Type Conflicts:** The same entity is represented using different data models/structures (e.g., an address is a single string in one DB, but a structured object in another).
    - **Domain Conflicts:** Different attribute values or formats for the same concept (e.g., salary in USD vs. EUR).
- **Techniques:** Involves identifying correspondences, resolving conflicts, and merging the schemas using techniques like view integration or superview/subview definitions.

Query Processing Issues in Heterogeneous Database

Query processing in HDBS is complex due to the autonomy and differences of the local systems.

1. **Query Translation:** The global query must be translated by the global processor into the canonical model, and then the wrappers must translate the resulting sub-queries into the native query language (e.g., SQL variant, OQL) of each LDBS.
2. **Data Type and Value Conversion:** Results from different LDBSs might use different data types or value formats (e.g., dates, currencies). The wrappers and global processor must perform necessary conversions before results can be combined.
3. **Semantic Heterogeneity:** Even if data is structurally the same, the underlying meaning may differ. (e.g., 'Employee' at one site means full-time staff, at another, it means anyone on the payroll). The global query plan must account for these semantic differences, often by using complex view definitions.
4. **Optimization Constraints:** The global optimizer has limited information about the internal workings/costs of the autonomous local systems, making global optimization much harder than in a purely homogeneous DDBS.